BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

Robust Intra-document Locations

Thomas A. Phelps and Robert Wilensky
University of California, Berkeley
phelps@cs.berkeley.edu, wilensky@cs.berkeley.edu

Abstract

Several types of existing and next-generation hypertext functionality, including external hyperlinks, annotations, and transclusions, rely on references to locations within another resource. If the document domain cannot guarantee referential integrity, but rather is more like the World Wide Web, in which documents change regularly and without notification, potentially invalidating internal location references, it is crucial to build robustness into the intra-document location resolution mechanism, so that locations continue to function even as documents change chaotically.

This paper aims to begin a process to evolve a standard for (normative) robust location descriptors and (non-normative) reattachment algorithms. We discuss criteria for evaluating the robustness of an intra-document location mechanism. Then we describe *robust locations*, an approach we believe meets these criteria. Robust locations include a standard minimal location descriptor and a recommended reattachment algorithm. We also suggest what can be done when the changes are so great that location resolution is problematic. Finally, we describe the implementation of robust locations within the Multivalent Document system.

Keywords: robust, location, annotation, Multivalent, hyperlink

1. Introduction

Hypertext research has long been concerned with the problem of the persistence of hyperlinks, that is, of dealing with problems that arise when one endpoint of a link, especially the destination, is unresolvable, because it was deleted, renamed, moved, or otherwise changed. A number of solutions have been proposed to deal with this problem when the reference is to a resource as a whole, including Handles [7], PURLs [11], URNs [17], ignoring it or requiring sychonous updates [4], W3Objects [5], web page tracking [9], agents [8], a "self-configuring information navigation infrastructure" [2] and Robust Hyperlinks [15]. This paper does not consider links between documents or "resources".

This paper does concern refer to "sub-resources", that is, fragments or locations within a target resource. "Location" is used in the sense that people use it, in terms of a "semantic" location that, after editing, can be recovered by scanning for the area of the document that most closely resembles the old content, as opposed to a definition based on geometric distances or character/word/paragraph counts. For example, in the World Wide Web, a URL can refer to a named anchor within an HTML document, and the XML Pointer Language (XPointer [19]) provides a general scheme for addressing internal XML document structures. However, in contrast to inter-document links, the issue of robust intra-document locations has received less attention. For example, most HTML browsers silently fail if they can locate the document but not the fragment within the document given by the URL.

We suggest that robust intra-document locations will become increasingly important: As the web develops as an increasingly sophisticated hypertext platform, many types of advanced functionality will emerge that gainfully exploit intra-document locations. For example, Nelson's transclusions [10] extract a *portion* of one document for display within another. Likewise, "stand-off" (or "out-of-line" or "out-of-band") annotation, in which annotations are stored separately from the annotated document, has numerous advantages over embedded annotation. Also, multimedia formats tend to come into existence independently from web authoring, and hence, any structuring must be imposed upon them externally, hence mandating sub-resource referencing.

At the same time as the need for intra-document location increases, the openness of the web will allow resources and locations to continue to change independently and chaotically. It is probably the case that resources will be modified with some frequency, even as the resource persists, hence creating the possibility that fragile sub-resource locations will fail to persist even if the persistence of resources is achieved.

2. Robustness Criteria

Fortunately, it is possible to make intra-document locations "robust" in uncontrolled, distributed environments, even when the target resource is not cooperating. By robust, we mean that one should be able to make an intra-document reference to a location within an arbitrary resource, save this description, and then re-establish the location in the future, after a document has undergone some class of mutations.

We use the term "location" as the object within a resource in need of positioning. We define locations as *indices into document content*. For example, for an HTML, XML, or plain text document, a location might be the point between two particular adjacent characters. If the document data format allows some other media element with addressable subcomponents, then the location might be the point before or after such a component, as for example the point between two frames of video. More complex locations may be built up out of primitive locations; for example, a span is the document fragment between two locations. (We do not consider locations of structural document objects, such as XML elements, per se, although we believe extension to such locations can be accommodated in a number of ways.)

To achieve robustness, two elements are needed: a *location descriptor*, which describes a location, and a *reattachment algorithm*, which attempts to reposition the descriptor within a possibly mutated target resource. We suggest that intra-document location descriptors and their associated robustness algorithms should provide a mechanism which is:

- *Robust to common changes in the referenced document.* Obviously this is the foremost requirement, without which the rest would be moot.
- *Gracefully degrading in the face of increasing change to the document.* Reattachment should proceed only if the computed location is likely to be the same location semantically as before the mutation. Hence minor changes should not pose much threat to reattachment; larger changes should cause reattachment to fail proportionally to the degree of change; if the change is too great, reattachment should fail.

  In information retrieval terms, the robustness algorithms need to minimize false positives, even at the cost of losing a small amount of accuracy (missing real, if greatly changed, positives). Thus, the reattachment algorithms should measure the likely quality of matches, and have a means to report failures to the user.

- *Based on document content.* For documents that change infrequently, and which have a fixed layout format, for example, those based on scanned page images, and perhaps, Adobe's PDF [1], it might be acceptable to rely upon geometric positions to indicate locations (although even here, if the document is expected to undergo editing in the future, it would be useful to anchor annotations to document content). Such an approach is certainly unsatisfactory for "flowed" document types, such as HTML and XML, because the page geometry depends on window size, font size, font family, and other factors that vary widely from viewer to viewer. (One annotation system that does work on web documents, Hot Off the Web [6], fixes the document dimensions, thus violating any number of usability guidelines.)

  Basing locations on document content gives a basis for robustness. Locations based on content will be resolvable independent of presentation characteristics. Moreover, if a document changes, reattachment algorithms based on finding "similar" content have some chance of finding the intended location even if a unrelated parts of the document change dramatically, and if relevant parts change only modestly.

- *Work with uncooperative servers.* A cooperative server could track and report all changes to its document collection, and notify interested parties when changes happen. Ideally, every server on the web would be cooperative, and speak the same versioning protocol. An even more cooperative server would manage reattachment as well: accepting references to its contents, storing them, and revising them whenever needed.

  Given that it is unlikely that even a small fraction of servers on the web will behave this way, a successful robustness strategy will need to husband all the data needed for the robustness algorithms within itself. On the other hand, once this requirement is met, the strategy will immediately operate on the universe of servers.

- *Extensible, to multimedia and various document types.* Documents can contain multimedia elements, such as images and video, and new document types are developed regularly. A robust location mechanism should extend naturally to accomodate these and new varied types in the future, without breaking existing locations.

- *Work with existing documents, servers, and clients.* Ideally, a solution would silently improve the existing world, but obviously any implementation has to involve at least servers or clients.

  At one extreme, the location descriptor could contain a complete edit trail of every character insertion and deletion, with timestamps, as in Nelson's Xanadu [10]. Or the descriptor could be a versioning history of edits in larger chunks and time periods. However, such descriptors would not be available even for all cooperating servers.

  It has been suggested [12] that each unit of a document can be made robustly addressable by giving each element a unique ID. However, we think it is unlikely that such IDs will soon become universal even within document types that support them. Even then, one could not use such a scheme to robustly identify locations within non-trival media elements, such as spans of text, much less within media types not supporting this particular addressing scheme.

- *Straightforward to implement.* We want robust locations to be widely supported, which is to say, implemented by many people on many computing platforms in many computer languages. We believe that simplicity and ease of implementation are important in this regard, as the widespread deployment of XML over the more complex SGML suggests.

- *Relatively small.* To be practical, locations should be a manageable size, relatively to documents, thus ruling out schemes that exploit complete edit trails, and so on, even if such were readily available.

Note that we have not added any criteria with respect to usability. This is because we expect that, to be effective, location descriptors will have to be automatically generated by clients, rather than hand-crafted by humans. Moreover, the details of the location descriptor should not concern most users.

3 Robust Locations

We now describe a strategy, called *robust locations*, to meet the above criteria. The core of the strategy is to (i) provide automatically generated location descriptors, which comprise multiple descriptions of a location, each of which captures different aspects of the document, and (ii) use heuristics when a descriptor does not resolve directly to a location to hypothesize the intended location, along with a measure of the degree of confidence in the hypothesized location.

Robust locations relies on the presumption of a document object model against which location descriptors are interpreted. We call this model the "simple, general document object model", or SGDOM. SGDOM is much simpler and more general than the XML or HTML DOM, in that it is meant to accommodate common aspects of a very wide class of possible document types. It is straightforward to map the primary structures of the XML or HTML documents onto this model, so clients specializing in these formats can create and interpret robust locations as well.

Briefly, the SGDOM describes a document as a tree of typed internal nodes (such as named XML elements), with terminal nodes being "media elements". The node types and the media elements are determined by the document type, according to a (hopefully well-defined) canonicalization process. Media elements are presumed to have a simple scalar index, whose meaning is determined by that media type.

For example, to interpret an HTML document according to the SGDOM for HTML, one follows the HTML parse tree, but splices out markup tags that merely serve to associated properties with spans of text, rather than truly structure content. That is, tags such as `B`, `A`, and `SPAN` are ignored for purposes of the model. In addition, nodes are added for text media types (so that an index into a text media element indicates the place before or after a character position).

The SGDOM allows nodes to have unique IDs, a la SGML/XML ID attribute types. However, there is no formal notion of attributes per se (or of processing instructions, or declarations, or other SGML/XML features) in the model.

3.1 Multiple Location Methods

Robust location descriptors describe locations redundantly, using a number of different data records. Each record type provides different quality-of-reattachment and robustness characteristics. Taken together, they provide considerable resilience.

Here we propose a core set of location descriptor records that are easy to generate, simple to understand and implement, and appear to work well in practice. Implementors are free to supplement these records with addition records, as described below.

1. A **Unique Identifier (UID)** is a name unique within the document, as per ID attributes in SGML/XML. These survive the most violent document changes, except its own deletion. As noted above, such identifiers will be available only through the foresight of the document author, and even then, only for locations corresponding to document elements, but not sub-element or non-element content.

2. A **Tree Walk** describes the path from the root of the document, through internal structural nodes, to a point within media content at a leaf.

   In practice, tree walks are the central component of robust locations. Since tree walks incrementally refine the structural position in the document as the walk proceeds from root to leaf, they are robust to deletions of content that defeat unique ID and context locations. Thus, tree walks are especially helpful for documents such as those that transclude dynamic content, as with stock quotes, where the content itself changes while the structural position remains constant.

   We describe tree walks with a sequence of node child numbers and associated node tags (generic identifiers), terminating with an offset into a media element. This is both a simpler, less expressive, and more redundant, representation than is allowed by XPointer. For example, consider the following tree walk into a particular HTML document:

   ```
   21/Professor/8 0/<TEXT> 0/ADDRESS 1/H3 0/BODY 0/HTML
   ```

   Reading right to left, the example can be decoded as starting at the root, taking its 0th child element, which should be labelled "HTML", then taking that node's the first child (numbered zero) to a node named "BODY", taking its second child, a node named "H3", taking its first child, "ADDRESS", then taking its first element, which should be a media leaf element labelled with its type, here "<TEXT>". The final portion of the descriptor, here `21/Professor/8`, is always medium-specific. We propose that TEXT can profitably be divided into words, as this provides robustness to any editing within other words. Thus, here the

`21/Professor` portion follows the pattern of the tree walk in implicitly treating its words as child nodes, such that the 22nd word should be "Professor". The `8` is then an offset eight chracters into this word, resolving to the point between the "o" and "r".

3. **Context** is a small amount of previous and following information from the document tree. We propose a context record containing a sequence of document content prior to the location, and a sequence of document content following the location. For example, for the location described by the tree walk above, let us suppose the word "Professor" was found in a sentence fragment that reads "congratulations on her promotion to Professor in the Computer Science Division". The context descriptor could be:

```
her+promotion+to+Professo    r+in+the+Computer+Science
```

with the previous and following context separated by a space, and metacharacters, including space, encoded with the same method as URL metacharacters. The amount of context to be saved can be arbitrarily large. (Here we use up 25 characters, which may be truncated at text element boundaries.)

Context records work well for short documents, which can be searched quickly. Moreover, in event of a failure, the saved context can be reported to the user as a facsimile of the former location. Without supporting location methods, however, simple context can easily be confused by the same pattern of words elsewhere in the document, a danger that grows larger the longer the document.

4 The Robust Location Reattachment Algorithm

Algorithms that exploit the multiple location descriptors can detect and correctly reattach many otherwise unresolvable references in the presence of a large class of changes. Here we present a "basic" reattachment algorithm, which reflects how we designed location descriptions to be used, and which we believe exploits them well. We do not claim this to be an optimal reattachment algorithm--many sensible variations of this algorithm, not to mention altogether different algorithms, are certainly possible, and determining the best algorithm requires empirical data we have yet to collect. Note, though, that the robust reference scheme does not require that user agents all implement the same reattachment algorithm.

Our algorithm first uses the unique identifier, then the tree walk descriptor, and then the context descriptor, continuing only if the previously tried methods are ineffective. We describe the details of each submethod below.

4.1 Unique ID Submethod

First, if the location has a UID, and the same UID is located in the document, the location is presumed to be resolved. This is because UIDs will presumably be used so as to not change their meaning via editing. Moreover, they will generally survive normal editing operations, unless the element itself is deleted. However, we presume that UIDs will generally only be available at key points in a document. (In theory, UIDs could be mechanically generated for every position, but this would only be possible for document formats that support them, SGML/XML but not ASCII or scanned images, and only if the source document can be modified. Moreover, automatic generation of UIDs would vitiate any useful semantic value that makes them privileged referrers in the first place.)

Failure to reattach via UID (that is, when there is no UID in the location, or the UID in the document has been deleted) causes control to yield to the tree walk submethod.

4.2 Tree Walk Submethod

In practice, the tree walk submethod is the workhorse reattachment method, due to the usual absence of a UID and the fact that its strong robustness properties are quite successful in resolving locations.

As shown in Figure 1, the tree walk exhibits strong natural robustness. In resolving the walk from root to leaf, changes of any magnitude within following sibling subtrees are safely contained. The walk is also invariant with non-structural changes to previous sibling trees.

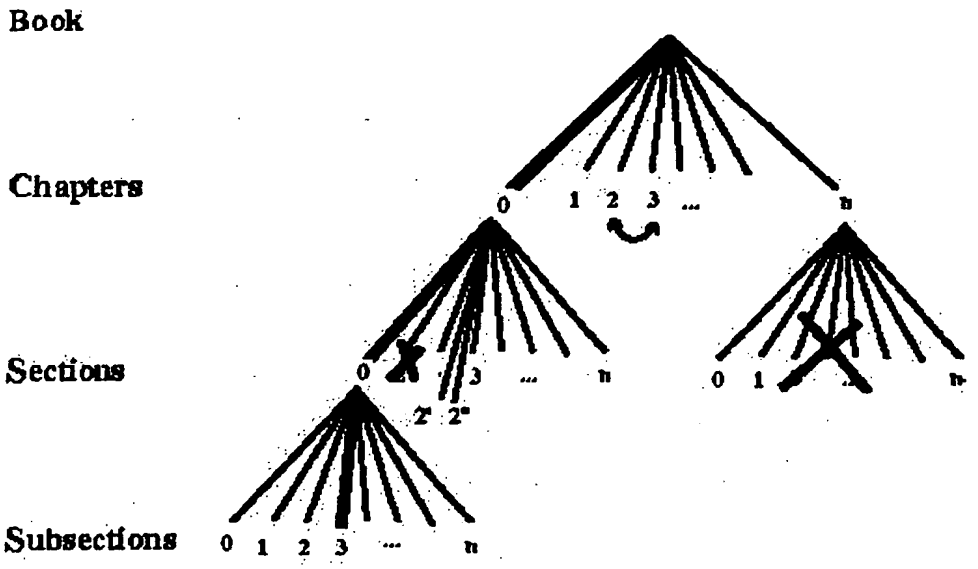

Figure 1. The tree walk method exhibits strong natural robustness. Areas that have been deleted, inserted, or interchanged in other subtrees may not even affect the desired tree path if the changed subtrees would be examined after the desired path in a depth-first traversal. Those at or after such such modifications may be recovered by the algorithm.

However, it is affected by structural changes to them. For example, introducing a new previous sibling, or removing one, can invalidate the walk. Tree walks can made much more robust with heuristics that repair broken paths. We check that a walk is valid by checking each node name, which is redundantly recorded in a robust location; for media indices, the general algorithm only checks that the indexed element can be found, after which control is passed to a medium-specific model that may have its own robustness methods.

Suppose that matching a tree walk of a robust location against the actual runtime tree fails at the exact saved offset (that is, that the walk is matched successfully up to the leaf, but that the leave is of the wrong type, or that the index of the media element is invalid). Then matching is tried on, first, the following sibling node, then the previous sibling, then the second following, the second previous and so on until the node names match. If the match fails previously, the same strategy is employed, with the proviso that the rest of the walk succeeds (possibly with additional approximations to exact matches). If no match can be found, we hypothesize that a new level of hierarchy was introduced, and skip a level of the tree. If still no match is found, we hypothesize that a level of hierarchy was removed, and skip the current location node/offset descriptor pair. If none of these attempts yields a match, the match attempt fails.

Each level of departure from the original description adds a weighted value to an overall "unreliability" factor for the match thus far. Moving to the following or previous sibling adds a small amount of uncertainty, with increasingly larger amounts as the siblings tested grow farther away, and skipping or assuming a level of hierarchy adds a larger amount. If the factor exceeds a client-defined bound, the tree walk match attempt fails.

This algorithm makes saved tree paths robust against any number and combination of sibling insertions, deletions and reorderings at any or numerous levels of the document tree. The search pattern prefers siblings closest to the saved location, rather than, say, searching from the parent's first child to its last, reasoning that the closest match is most likely the best match. This search tactic has the desirable algorithmic property that the first match found is also the closest.

There are a number of circumstances in which the tree walk submethod will perform a questionable realignment. For example, it may happen that a numerical offset now refers to a new node of exactly the same name as the original, in which case a match would be reported at that point. However, it is likely that such a coincidence will not occur for the remainder of the subtree rooted at that node, and this false match will be caught, and another match attempted. (If the entirety new tree is identical to the original, the match will succeed without any penalty, a situation with some may consider as a flaw, however unlikely this is in practice.)

Media elements may marshall their own robustness tactics. For example, as noted above, we segment textual media elements into words, and a tree walk locator may refer to a point between characters within a word. For the purposes of resolving such references, we can treat the words just as if they are element names, and just continue to apply the same tree walk resolution method. For example, suppose one attempts to resolve such a location in a situation in which the

word prior to the one containing the location has been deleted. Assuming that the word following the original target word was not the same, the tree walk now becomes invalidated, because the word asserted to be in a given position is not the same as the one that is there. However, the progressively widening searching heuristic will quickly find the target word in its new position.

4.3 Context Submethod

If the structural document tree has been changed too much for the tree walk's tactics to recover--perhaps a sentence has been cut from one chapter and pasted into another far away--re-registration is attempted using the context data record.

As with the tree walk, the closest match to the original position is the preferred. A search is done forward and backward, with the nearer match chosen. If neither direction matches, more and more of the context is shed until a match is found, or until the length of the string used for the search drops below a threshold.

The initial search position is set from the furthest extent that the tree walk described above could be resolved. This sets a restricted domain within which to search, which benefits proximity matching and performance. If no match is found within a subtree, the search climbs up the tree by one node and tries again within that subtree until a match is found or it has climbed to the root. Within a subtree, leaves are searched left to right.

As with tree walk matching, an overall unreliability factor for the match is maintained. As the search examines ever larger subtrees, the unreliability is increased. If some but not all of the context is matched, unreliability is increased.

Approximate string matching, while not presently implemented, should prove invaluable in matching context, as it is robust to the small changes typically found in spelling corrections.

4.4 Match Outcomes

At the end of the match, several results are possible: The match may succeed without the use of reattachment heuristics; it may succeed using the heuristics, with a small unreliability factor; it may succeed with a large unreliability factor; or, it may fail (that is, the reliability factor is too large--infinite if there is no match at all).

Where reattachment fails, we require that clients present this fact to users. Clients may have different policies for handling the other cases. For instance, a client may not inform the user at all of what it considers to be minor reattachements; it may flag reattachments considered somewhat unreliable, perhaps asking the user to verify the repositioned object.

4.5 Meeting the Robustness Criteria

Robustness is achieved within and among the submethods of unique ID, tree walk, and context. The methods elaborated above provide for graceful degrading in performance with increasing change to the document. (We describe some limited empirical experience below.) Also, robust locations are based on document content, as required.

Although the implementation details of digital document systems differ in the ease of implemention of different location types, the three given are probably among the easiest across a range of systems. Within HTML/XML clients, unique IDs are in principal already available, as some variation is used for intra-document anchors. With increasing support for the Document Object Model (DOM) [19], tree walks are available to scripts in various programming languages. If tree navigation is available, then context is available with a search.

Location descriptors are self contained, needing nothing from a server beyond the document to which it refers. Their records can easily be stored separately from the document to which they refer.

Locations can be managed entirely at the client, and an implementation there would immediately bring the benefits to all existing documents and servers.

As the example below shows, an individual location record, which concatenates these three subtypes, is small; in the current implementation, each generally costs about 50-100 bytes.

Although the location descriptor has not been extended to many multimedia data types, this need has been prepared for. A tree walk should be successful to the degree to which the new data type is structured. If nothing else, the final element in the walk is the offset into the leaf. Text uses it as a character offset, but other media types could encode medium-specific internal coordinates.

As well as new media types, the system is open to new location submethods, as for instance the "minimal unique string" method used by the ComMentor annotation system [16]. While too complicated to mandate for a minimal standard, this system uses a Patricia tree data structure to identify locations with the smallest unique string in the document, and although it seems that the addition of a single character could render this location non-unique, when coupled with an initial character offset, it reportedly works well in practice. This method could be added in parallel to the unique ID, tree walk and context methods, or perhaps as the way context is computed, with the mandate that systems must ignore methods they don't understand, just as web browsers ignore tags they do not recognize.

5. Implementation

We have implemented robust locations within the Multivalent Document system [14] [18]. Our implementation lacks application to a variety of media types and testing by a large user community, but otherwise has proven robust beyond expectation.

5.1 Example

Below is an example containing a number of robust locations. In the Multivalent system, a hub document describes a number of *layers* and *behaviors* which the client assembles into a document. One of the layers is a base layer, from which which the primary document representation is constructed. The other layers and behaviors may provide arbitrary functionality, including, but not restricted to, various forms of annotation. Since the layers and behaviors are all potentially separate resources, a hub document may be used to describe stand-off annotations on a document.

Such is the case in our example, Figure 2, which is presented as follows by a Multivalent client as follows:

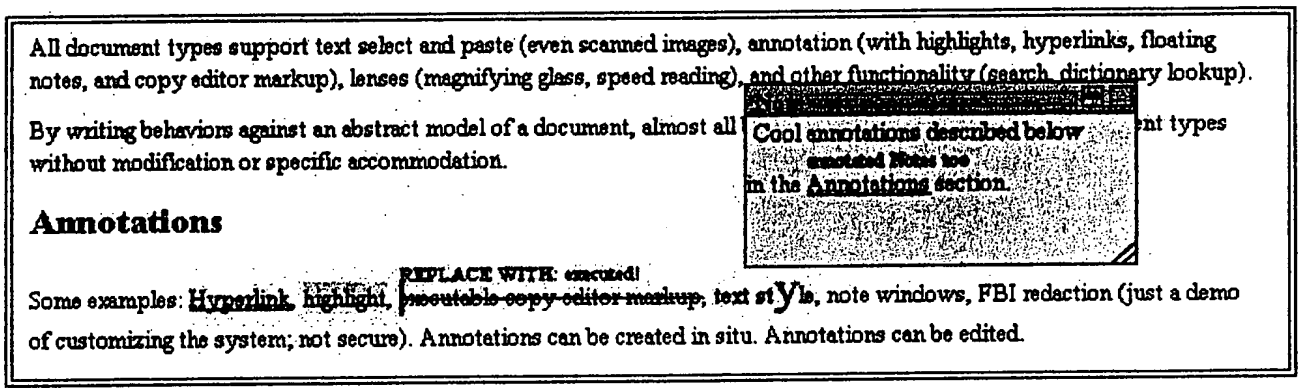

Figure 2. Some stand-off annotations on an HTML document

In this example, three annotations appear on an HTML page: A PostIt-style note, a "REPLACEMENT" copyedit annotation on the main document text, and "COMMENT" copyedit annotation on the text in the note. Each of the annotations is separate from the HTML page, and so the two copyedit annotations refer to their targets via robust locations. In particular, each of these is an instance of a large class of "span" annotations, each of which has a starting and ending location. So, each of these annotations uses a pair of robust locations. (The note is positioned geometrically, and so does not use robust locations. However, one could attach a note, or some non-span annotation, to some base document text, in which case robust locations could also be used.)

The presentation is created by the Multivaleant client from the following hub document (Figure 3). The hub is itself an XML document. The "Span" elements each describes one of the two copyeditor annotations. Each Span has a "Start" and a "End" element, each of which contains a robust location. Each location data item component is encoded by a separate attribute, namely, UID, TREE, and CONTEXT, which are shown in italics. (There are no UIDs in the example, as there were none in the target document, as is typical.)

```
<MULTIVALENT  URL="systemresource:/sys/splash/Multivalent.html"
  SEARCHNB="ON"  GENRE="HTML">

<Layer  NAME="Personal"  BEHAVIOR="multivalent.Layer"  URL="inline">

<Span  BEHAVIOR="multivalent.std.span.ReplaceWithSpan"  CREATEDAT="942341857100"
  NB="COPYEDNB"  CONTENT="executable+copy+editormarkup"  LENGTH="29"
  INSERT="executed">
```

```
<Start  BEHAVIOR="multivalent.Location"
  TREE="4/executable/0 0/<TEXT> 11/P 2/BODY 0/HTML"  CONTEXT="highlight+exec utable+copy">
</Start>
<End  BEHAVIOR="multivalent.Location"
  TREE="7/markup/6 0/<TEXT> 11/P 2/BODY 0/HTML"  CONTEXT="editor+markup text>
</End>
</Span>

<Note  NAME="NOTE1011000469"  BEHAVIOR="Note"  X="300"  Y="100"  WIDTH="193"  HEIGHT="98"  POSTED="T
Cool annotations described below in the Annotations section.</Note>

<Span  BEHAVIOR="multivalent.std.span.AwkSpan"  CREATEDAT="942342203130"
  CONTENT="AnnotationsAnnotations"  NB="COPYEDNB"  LENGTH="12"
  COMMENT="annotated+Notes+too"  ROOT="NOTE1011000469">
<Start  BEHAVIOR="multivalent.Location"
  TREE="10/Annotations/0 0/<TEXT> 0/LINE 0/NOTE 0/CONTENT"  CONTEXT="the Annotation+section.">
</Start>
<End  BEHAVIOR="multivalent.Location"
  TREE="10/Annotations/11 0/<TEXT> 0/LINE 0/NOTE 0/CONTENT"  CONTEXT="the+Annotations section.">
</End>
</Span>

</Layer>
</MULTIVALENT>
```

Figure 3. A "hub" document, which persistently captures some annotations and their corresponding robust location descriptors.

Note that the spans have augmented the robust locations by adding a CONTENT attribute that contains the textual content along the extent of the span. If a failure happens in one of its endpoints, the other can be estimated by taking an offset the length of the CONTENT from the successful one. In addition, the second span augments the basic location resolution mechanism by anchoring to a secondary tree root, that for the Post-it note.

This hub document was itself generated by the user first opening some document, initially the underlying base document, and then using a Multivalent client to create annotations. The hub document is generated upon saving. Thus, the entire hub document, including the robust locations, was generated automatically.

5.2 Failed Reattachments

The screen dump in Figure 4 below shows one strategy we have implemented for handling failed reattachments. It presents to the user a floating note listing unattachable spans, which have been reconstituted within the note from the saved context, to which the locations, here paired in a span annotation, are placed.

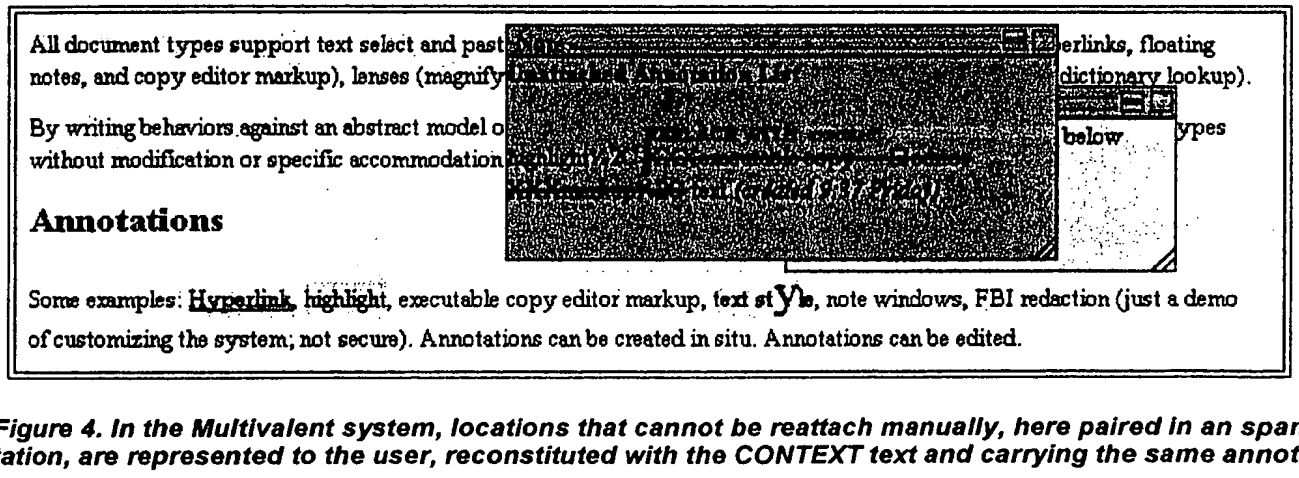

Figure 4. In the Multivalent system, locations that cannot be reattach manually, here paired in an span annotation, are represented to the user, reconstituted with the CONTEXT text and carrying the same annotation.

The next screen dump, Figure 5, shows an associated user interface we have implemented to allow the user to manually reposition an un-attachable span. The user first makes a selection, whose end points will serve as the new start and end points of the annotation, moves the cursor into the unattachable span in the note, clicks the mouse, and finally, chooses `Morph to Current Selection` to move the reconstituted span to its new location.

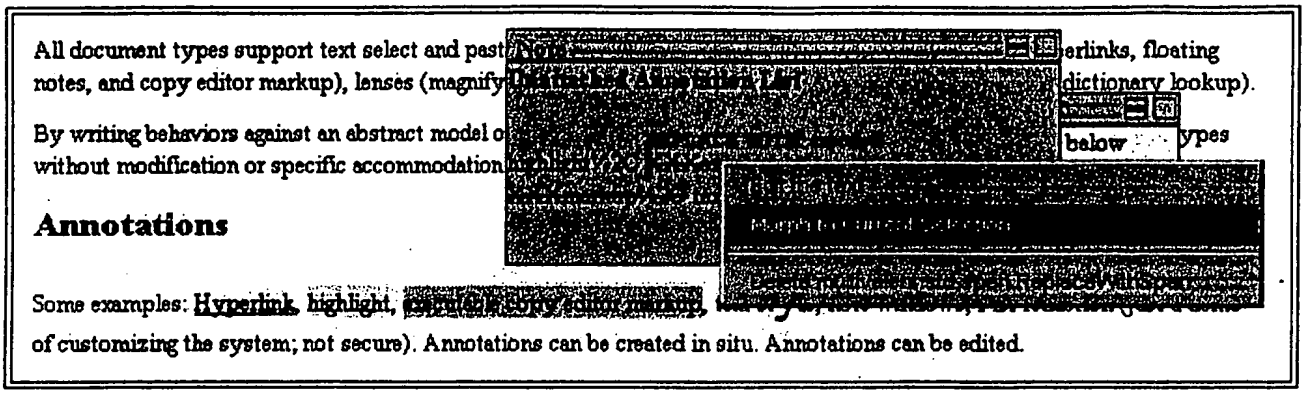

Figure 5. The user can reassociate a span annotation by selecting the correct location and making a choice from a pop-up context menu in the reconstituted span.

5.3 Testing

Properly testing robust locations requires real use by a user community. Our Multivalent client is still a prototype, so such studies have not yet been undertaken. We offer the following relevant pieces of evidence that the algorithm will work well in practice, acknowledging that they are no substitute for actual measurement.

We collected some data using a prototype, less sophisticated implementation of robust locations in TkMan [13], a browser for UNIX manual pages that has been in use by a sizeable community for some time. Specifically, of 754 annotations that needed repositioning because the referenced man pages underwent change out of control of the annotator, 742 annotations were automatically repositioned, leaving 12 to be reapplied by the user. In most of the latter cases, the associated position had in fact been deleted entirely.

We tested the implementation described here by creating a number of robust locations for some documents we manage, and then mutating the documents in various ways. The algorithm performed extremely well. Indeed, to generate the example of unattachable links shown above, we tried to make the algorithm fail by repeated modifications of the base document, but gave up in frustration, as a different robustness submethod would continually correctly identify the proper location. Instead, we eventually broke the location descriptor itself manually.

Perhaps more compelling is our use of this robust location implementation on a number of documents which were not under our control. In one case, we posted some annotations on the DARPA home page, which we left unchanged for over one year. During this time, each annotation reattached correctly through many alterations of the page, including one in which the page was apparently imported into an HTML authoring tool, which placed the entire document content into cells of a newly introduced table. (The location finally broke when a major website redesign using frames rendered the location to the page completely meaningless).

6. Related Work

6.1 Webvise

Webvise [3] is an open hypermedia service that provides a variety of services including "the ability for users collaboratively to create links from parts of HTML Web pages they do not own and support for creating links to parts of Web pages without writing HTML target tag" and "supports certain repairs of links that breaks due to modified Web pages".

Webvise proposes location specifications ("locSpecs") for a variety of media, and redundantly stores locSpecs. For instance, for a span of text, the locSpec attributes are:

- Reference: a bookmark ID or a HTML target name
- Selection: the text of the span to search for
- Selection context: some surrounding text
- Axis: a position, such as start position, and length of span on a character axis

Thus, Webvise and our robust location proposal share some goals and approaches. Webvise describes locSpecs for

several media types, such as database records, not considered here.

The present work adds to the Webvise work a tree walk location method. As suggested above, the tree walk is essential for live transclusions, and has the desirable properties of working with any structured media type, working on partially constructed trees (subtrees can be faulted in on demand), being robust to user actions that can produce large changes in the document such as swapping chapters, and, even when it does fail, providing a more focussed domain for a context search. We also introduce a set reattachment algorithms, with an explicit relationship among them, and the use of an internal measure of the amount of divergence from the saved specification. Finally, the present work proposes a user interface for dealing felicitously with locations that could not be reattached automatically.

6.2 ComMentor

ComMentor [16], which annotates HTML pages, records the shortest unique substring (based on string position trees, implemented as Patricia trees) and redundant context.

Evaluated against our criteria, ComMentor has a number of drawbacks. The computation of the Patricia tree data structure is complex, at least compared to the proposed methods. The computation does not scale up, as the entire document must be processed to guarantee uniqueness, whereas, for example, the tree walk need not even instantiate a subtree that it does not traverse, prehaps faulting in that subtree on demand. If locations are made on documents in progress, such as those used by annotations on documents being collaboratively written, unique string may be relatively short when the document is short, but then lose their uniqueness as the document grows, leading to failure. If the reattachment fails, degradation is not graceful, as there is no way choose among possible matches. It is not easy to see how Patricia trees generalize to multimedia content.

6.3 XPointer/XLink

The task of marking a location and then recovering it in the future in the face of change seems to be outside the domain of XPointer [19]. Given that XLink [19] uses XPointer as its locator mechanism, out-of-line XLink links are not robust.

XPointer describes many means for traversing a document tree, including node position, and id and attribute values. If a program knows where it is going, it can get there with an appropriate means. However, XPointer leaves to the client the choice of what to do if a previously stored tree path is no longer available in the current runtime tree. In fact, XPointer and the present work do not conflict: XPointer concerns tree traversal, and the present work monitors the traversal for correctness and takes recovery action when the path diverges from what is expected.

6.4 Microcosm

The Microcosm system [4] can piggyback on top of other systems, such as Microsoft Word, and augment it with new or improved hypertext functionality.

The authors of Microcosm suggest that, although the "editing problem" (as they call it) "is probably a major flaw in the Microcosm model when it comes to scaling up to deal with globally distributed documents", in their experience, documents do not change frequently enough that the lack of a solution is debilitating. According to their analysis (page 90):

- Typically, around 50% of all links have source anchors which are generic [queries] or local. Documents contaning such links may be freely edited without causing any problems.
- Typically about 50% of destination anchors are the "top of the document" [geometric]. Again, such documents may be edited without causing problems.
- Documents consisting of bitmapped pictures, videos and sound are very rarely edited, and even when they are, it is very rarely in such a way as to alter offsets of objects within them.
- Documents such as object drawing and spreadsheets, which use the name of an object [a unique ID] rather than some offset to identify an anchor, are not subject to editing problems, so long as the edit does not change the object's name.

Nevertheless, some documents are problematic, and Microcosm does not offer good solutions for handling them. One of their proposed approach prevents mutable documents from participating in links, and the other approach requires that documents be editing together with all their links.

7. Future Work

The proposed mechanisms seem to work well in limited use, but they need to be tested with a wider public. Some refinement of the heuristic weightings requires measurement of the typical kinds of locations needed by users in everyday

work, and measurement of the changes documents undergo. For example, short context strings worked well in our limited experimentation, but we suspect that longer strings would be better in extended applications, and increasing the amount of context can be done inexpensively.

The proposed mechanisms have not been tested on multimedia data types very much. Microcosm argues that non-text type do not change frequently. Webvise seems to have made a good start at defining location methods for a number of diverse media types.

Many alternative reattachment algorithms are possible, ranging from variations of the one we propose (for example, using approximate string matching rather than exact string match) to entirely different mechanisms. Comparisons of such algorithms would be most useful.

Here we encoded robust locations as a set of element attributes. It might be useful to include robust intra-document locations in URLs as well. To do so, one could device a syntax to allow robust locations to extend URLs, as per XPointer (perhaps by extending XPointer syntax). Alternatively, one could shoehorn robust locations into the current URL syntax, analogously to the way we propose doing so for robust URLs in [15].

8. Conclusion

The examples of Webvise, Microcosm, ComMentor, XLink, Multivalent Documents, and others suggest an increasing need for robust intra-document locations. There appears to be incipient agreement in some areas: A unique ID is a good primary descriptor, if it exists. Several authors have found a context fragment to be useful. We make the case for adding tree walks, and for algorithms that combine all these sources.

Therefore, we think it appropriate that the next step be a web working draft that defines the robust location specification so that makers of web browsers and tools can begin generating robust location descriptors. We believe this process should prove relatively straightforward, even if robustness algorithms may take more time to perfect and implement.

Acknowledgements

This research was supported by Digital Libraries Initiative, under grant NSF CA98-17353.

Appendix: Comparison of Algorithms

The following table summarizes the types of robustness of locations.

Method	Type	Robust to change type	Backoff tactic	Quality of reattachment
Unique Identifier	Core method	any except deletion of itself	n/a	Perfect
		deletion	none	n/a
Tree path		insert/delete/reorder siblings	search siblings of decreasing propinquity	Gracefully degrades
		add/remove level of hierarchy	skip or pseudo match level	
Context		location moved anywhere	search within matched tree path, prefer closest	
		move and context changed	match less contest, increasing unreliability	
		move and context changed	fuzzy/approximate string match (unimplemented)	
(Failure)	Implementation dependent	n/a	Manual reattachment	n/a
Span (built on top of base	Extension	loss of one endpoint	Attach missing endpoint at fixed offset from the other	Gracefully degrades

types)				
(Media-specific extensions)		(medium specific)	(medium specific)	(medium specific)

References

[1] Adobe Systems. Acrobat 4.0 (http://www.adobe.com/products/acrobat/main.html).

[2] Paul Francis, Takashi Kambayashi, Shin-ya Sato and Susumu Shimizu. Ingrid: A Self-Configuring Information Navigation Infrastructure. December 11-14, 1995. (http://www.ingrid.org/francis/www4/Overview.html)

[3] Kaj Grønbæk, Lennert Sloth, Peter Ørbæk. Webvise: Browser and Proxy Support for Open Hypermedia Structuring Mechanisms on the WWW. Proceedings of the Eighth World Wide Web Conference 1999, Toronto, Canada (http://www8.org/w8-papers/3a-search-query/webvise/webvise.html).

[4] Wendy Hall, Hugh Davis and Gerard Hutchings. *Rethinking Hypertext: A Microcosm Aproach*, Kluwer Academic Publishers, 1996.

[5] David Ingham, Steve Caughey, Mark Little. Fixing the "Broken-Link" Problem: The W3Objects Approach. Computing Networks & ISDN Systems, Vol. 28, No. 7-11, pp. 1255-1268: Proceedings of the Fifth International World Wide Web Conference, Paris, France, 6-10 May 1996 (http://arjuna.ncl.ac.uk/group/papers/p050.html).

[6] Insight Development. Hot Off the Web software (http://www.hotofftheweb.com/).

[7] R. Kahn and R. Wilensky. A Framework for Distributed Digital Object Services. cnri.dlib/tn95-01, May 13, 1995 (http://www.cnri.reston.va.us/k-w.html).

[8] Sofus Macskassy and Leon Shklar. Maintaining information resources. *Proceedings of the Third International Workshop on Next Generation Information Technologies* (NGITS'97), June 30-July 3, 1997, Neve Ilan, Israel (http://www.cs.rutgers.edu/~shklar/papers/ngits97/).

[9] Mind-it. (http://www.netmind.com/html/individual.html)

[10] Theodor H. Nelson. Computer Lib/Dream Machines. 1974. Also see the Xanadu home page (http://www.xanadu.net/).

[11] OCLC PURL Service (http://www.purl.org).

[12] Frank Olken. Private communication.

[13] Thomas A. Phelps. TkMan: A Man Born Again. The X Resource, 1(10):33-46, 1994.

[14] Thomas A. Phelps. Multivalent Documents: Anytime, Anywhere, Any Type, Every Way User-Improvable Digital Documents and Systems. Ph.D. Dissertation, University of California, Berkeley. UC Berkeley Division of Computer Science Technical Report No. UCB/CSD-98-1026, December 1998. Also see the general and technical home pages (http://www.cs.berkeley.edu/~phelps/papers/dissertation-abstract.html, http://www.cs.berkeley.edu/~wilensky/MVD.html, http://www.cs.berkeley.edu/~phelps/Multivalent/).

[15] Thomas A. Phelps and Robert Wilensky. 2000. "Robust Hyperlinks Cost Just Five Words Each", University of California, Berkeley Computer Science Technical Report, CSD-00-1091 (http://www.cs.berkeley.edu/~wilensky/robust-hyperlinks.html).

[16] Martin Roscheisen, Christian Mogensen and Terry Winograd. Beyond Browsing: Shared Comments, SOAPs, Trails, and On-line Communities. Proceedings of the Third World Wide Web Conference: Technology, Tools and Applications, Darmstadt, Germany, April, 1995.

[17] K. Sollins and L. Masinter. Functional Requirements for Uniform Resource Names, Network Working Group Request for Comments 1737, December 1994 (http://www.w3.org/Addressing/rfc1737.txt).

[18] Robert Wilensky and Thomas A. Phelps. "Multivalent Documents: A New Model for Digital Documents", University

of California, Berkeley Computer Science Technical Report, CSD-98-999, March 13, 1998 (http://www.cs.berkeley.edu/~phelps/papers/techrep98-abstract.html).

[19] World Wide Web Consortium. Structured Vector Graphics (SVG), XPointer, XLink, Document Object Model (DOM) (http://www.w3.org).

Vitae

Tom Phelps earned his Ph.D. in Computer Science in 1998 from the University of California, Berkeley.

Robert Wilensky is a Professor at the University of California, Berkeley. He is Principal Investigator of UC Berkeley's Digital Library Initiative Project.